Jurnal Computech & Bisnis, Vol. 1, No. 2, Desember 2007, 90-104
ISSN 1978-9629

PERBANDINGAN KOMPRESI *FILE* MENGGUNAKAN ALGORITMA *RUN LENGTH* DENGAN *TWO LEVEL HOSHING*

Rohmat Nur Ibrahim
STMIK Mardira Indonesia, Bandung

Abstract

Problems faced by users of information technology one of which is the size of large files that need a large storage media, and required a long time to save it. By compressing a large file is a solution to save time and storage media. Compression system was extensive but because of the limitations, the theme of which will be described on this occasion only covers problems in the system comparison of the results of the compression of a file into the software by using algorithms Run Length and Two Level Hoshing. With the aim to build software that can support the compression system, does not always have to provide large storage media and time efficiency.

***Keywords**: algoritma Run-Length, Two-Level-Hoshing, Compression, Decompression*

Abstrak

Permasalahan yang dihadapi oleh pengguna teknologi informasi salah satunya adalah ukuran file yang besar sehingga diperlukan suatu media penyimpanan yang besar serta diperlukan waktu yang cukup lama untuk menyimpannya. Dengan mengkompres suatu file yang berukuran besar merupakan solusi untuk menghemat media penyimpanan dan waktu. Sistem kompresi sangatlah luas tetapi karena keterbatasan, maka tema yang akan diuraikan pada kesempatan ini hanya meliputi permasalahan dalam sistem perbandingan hasil kompresi suatu file ke dalam perangkat lunak dengan menggunakan Algortima *Run Length* dan *Two Level Hoshing.* Dengan tujuan dapat membangun perangkat lunak yang dapat menunjang sistem kompresi tersebut, tidak selalu harus menyediakan media penyimpanan yang berukuran besar dan efesiensi waktu.

Kata Kunci: *algoritma Run-Length, Two-Level-Hoshing, Compression, Decompression*

PENDAHULUAN

Kemajuan teknologi di bidang statistika, multimedia, seni, photografi, dan bidang-bidang lain yang membutuhkan penyimpanan *File* informasi dalam bentuk laporan-laporan, gambar, foto baik itu *File* gambar diam atau bergerak tentunya sangat membutuhkan ruang penyimpanan yang relatif besar. Peningkatan kebutuhan tersebut ternyata berakibat sangat mahal bagi media penyimpanan dan media transmisi. *File* seperti di atas memiliki ukuran yang sangat besar sehingga menyita tempat yang cukup banyak di dalam memori

dan membutuhkan *bandwidth* yang besar untuk dapat mentransmisikannya dengan baik. Untuk itu *File* tersebut harus dikompresi untuk menghemat ruang penyimpanan dan *bandwidth*. Teknik kompresi dapat dikategorikan kedalam dua tipe algoritma dasar yaitu kompresi *Run-Length* dan *Two-Level-Hoshing*. Yang mana kedua algoritma kompresi tersebut dapat membangun *File* aslinya tanpa kehilangan informasi atau kualitasnya, meskipun rasio kompresi yang dihasilkan begitu besar. Untuk menghasilkan kompresi yang baik, beberapa distorsi harus diabaikan dalam pembangunan *File* kompresi. Algoritma kompresi *Run-Length* dan *Two-Level-Hosing* dapat menghasilkan rasio kompresi yang masing-masing memiliki kelebihan dan kekurangan.

Pada masa sekarang ini sangat dibutuhkan suatu teknik pemampatan *File* dalam format (extension) doc., cdr., bmp., JPEG., dat., txt dan lain-lain yang ukuran *File*nya berbeda-beda. Untuk itu diperlukan sebuah sistem yang dapat menyimpan *File* dengan ukuran yang kecil dan dapat disimpan dalam media penyimpanan yang berukuran kecil misalnya disket.

PEMBAHASAN

Kompresi **(Compression)** adalah reduksi atau mengecilkan ukuran file data agar menghemat ruang penyimpanan file dan memperpendek waktu transfer file. Teknik pemampatan dengan tujuan memperkecil ukuran file tanpa harus menghilangkan informasi-informasi penting yang ada didalamnya. Dekompresi **(Decompression)** adalah sistem yang digunakan untuk mengembalikan sebuah file yang telah mengalami pemampatan (*compression*) pada bentuk dan ukuran file aslinya sehingga isi dari file tersebut dapat dilihat kembali. *Lossy compression* adalah jenis algoritma kompresi yang menghasilkan data yang terkompresi yang berkurang kualitasnya dari data asli. Teknik ini teknik yang *irreversible* artinya citra yang dihasilkan tidak sama seperti citra aslinya karena adanya distorsi yang diabaikan. Rasio kompresi yang dihasilkan biasanya lebih besar dibandingkan kompresi *lossless.Lossless compression* adalah algoritma yang digunakan mengkodekan data secara sempurna, tanpa kehilangan informasi atau data yang hilang dalam pengkodeannya. Apabila didekodekan, citra yang dihasilkan sama seperti citra aslinya. Rasio kompresi yang dihasilkan biasanya tidak begitu besar. Pemampatan File dapat bersifat “Lossy” seperti lossy JPEG atau “Lossless” seperti TIFFL-ZW. Pemampatan Lossy memungkinkan pemampatan data lebih kecil daripada jenis Lossless.

Dalam sistem kompresi file ada berbagai algoritma yang dapat digunakan, antara lain sistem kompresi file dengan menggunakan algoritma *Run-Length* algoritma *Run-Length* memiliki kelebihan tersendiri khususnya jika digunakan dalam proses kompresi maupun proses dekompresi karena sebagian besar hasil presentasi dari proses kompresi dengan menggunakan algoritma ini hasilya sangat baik khususnya untuk proses kompresi file yang berbentuk grafis karena file grafis berekstensi JPEG, bmp, cdr dan lain lain dimana file ini memiliki karakter yang berderet lebih dari tiga karakter namun algoritma ini juga dapat mengkompres file yang berekstensi txt, doc, dat dan lain-lain. Proses kompresi pada karakter yang berulang-ulang kita dapat menggunakan Algoritma *Run-Length*, dimana pada saat karakter yang sama diterima secara berderet lebih dari tiga algoritma ini mengkompres file tiga karakter yang berderetan tersebut, sehingga Algoritma ini sangat efektif untuk file-file dalam bentuk grafis dimana biasanya berisi deretan karakter panjang yang sama, seperti contoh berikut ini:Suatu karakter ASCII deretan 8 *bit* lalu diikuti oleh sebuah *bit* yang memberikan infomasi jumlah karakter yang dikompres seperti pada karakter BBBBBBBB atau dalam biner

00001011 sebanyak 8 kali, sehingga hasil yang didapatkan dari proses kompresi algoritma *Run-Length* file tersebut dapat dikompres sehingga jumlah atau ukuran file-nya menjadi lebih kecil dari file aslinya sebelum di lakukan proses kompresi.

Untuk membandingkan sistem kompresi Algoritma *Run-Length* dengan metode pembanding yaitu algoritma *Two-Level-Hoshing*. Algoritma *Two-Level-Hoshing* digunakan dalam sistem kompresi file yang mempunyai empat *bit* sebelah kiri yang secara berurutan sama terutama dalam file-file yang berektensi txt, doc, dat, dan lain-lain namun tidak menutup kemungkinan algoritma ini juga dapat digunakan dalam proses kompresi untuk file-file yang berekstensi JPEG, bmp, cdr. Secara garis besar proses kompresi Algoritma *Two-Level-Hoshing* yaitu mengkompres empat *bit* pertamanya sama diterima secara berderetan sebanyak tujuh kali atau lebih, Algoritma ini memampatkan data tersebut dengan *bit* penanda yang kemudian karakter pertama dari deretan empat *bit* yang sama diikuti pasangan empat *bit* terakhir deretan berikutnya dan ditutup dengan *bit* penutup. Algoritma *Two-Level-Hoshing* ini sangat efektif pada file-file yang berbentuk text.

Ketika karakter yang empat *bit* pertamanya sama diterima secara berderetan tujuh kali atau lebih, maka Algorima *Two-Level-Hoshing* akan memampatkan data tersebut dengan *bit* penanda, Kemudian karakter pertama dari deretan empat *bit* yang sama diikuti dengan pasangan empat *bit* penutup.

Contohnya adalah teks berisi tulisan "mengangkat" yang bila kita terjemahkan dalam *heksadesimal* dan *biner* adalah sebagai berikut:

Tabel Hexadesimal dan Biner

Karakter	Heksadesimal	Biner
m	6D	01101101
e	65	01100101
n	6E	01101110
g	67	01100111
a	61	01100001
n	6E	01101110
g	67	01100111
k	6B	01101011
a	61	01100001
t	74	01110100

Apabila kita perhatikan karakter-karakter tersebut di atas memiliki empat *bit* sebelah kiri yang sama, yaitu 0110, Karakter seperti inilah yang di kompresi oleh algoritma *Two–Level–Hoshing*, Saat karakter yang empat *bit* pertamanya sama di terima secara berderetan sebanyak tujuh kali atau lebih, algoritma ini memampatkan data tersebut dengan *bit* penanda yang kemudian karakter pertama dari deretan empat *bit* yang sama diikuti pasangan empat *bit* terakhir deretan berikutnya dan di tutup dengan *bit* penutup.

ANALISIS SISTEM

1. Algoritma Kompresi (Compression) dengan Metode Run-Length

Proses Kompresi pada Algoritma *Run-Lenght* adalah dengan cara mencari karakter yang berulang-ulang lebih dari 3 (tiga) kali pada suatu *File* yang kemudian akan diubah menjadi sebuah *bit* penanda, yaitu deretan 6 *bit* yang membentuk suatu karakter ASCII lalu diikuti oleh sebuah *bit* yang memberikan informasi jumlah karakter yang berulang kemudian ditutup dengan karakter yang dikompres seperti pada karakter A atau 100001 sebanyak 8 kali, hal ini diharapkan hasil Kompresi/Dekompresi yang diperoleh hasil yang maksimal tanpa menghilangkan atau mengurangi file yang ada, kelebihan dan kekurangan dalam proses Kompresi/Dekompresi pada kedua Algoritma ini yaitu Algoritma *Run-Leng* dan *Two-Level-Hoshing* merupakan perbandingan yang

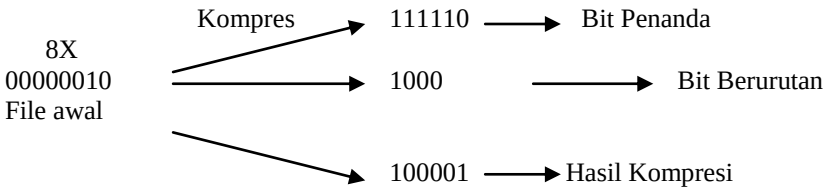
akan memberikan solusi bagi *User* (pengguna) dalam memilih Proses mana yang akan digunakan , contoh kasus tersebut adalah sebagai berikut:

Dik :
Karakter A = 65 (ASCII)
Dalam ASCII A = 100001
Bit Penanda = 111110

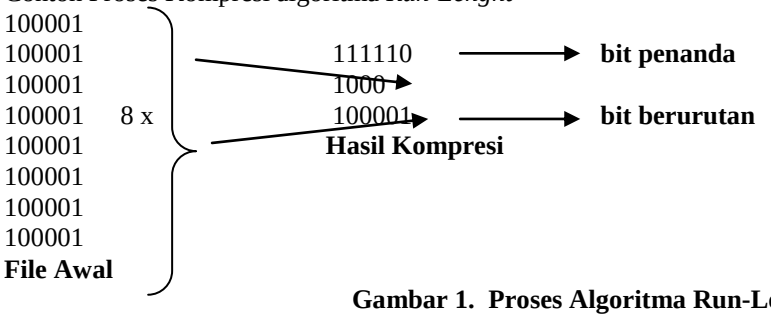
Dit : *kompresi Run-Lenght*...?

Jwb : cari karakter yang sama secara berurutan dari 3 (tiga) karakter yaitu : 100001

Karakter = ASCII
A =65 = 100001



Contoh Proses Kompresi algoritma *Run-Lenght*



Gambar 1. Proses Algoritma Run-Length

Pada saat kita akan melakukan pemampatan/kompresi terhadap sebuah *File*, maka terlebih dahulu dilakukan tinjauan apakah terdapat deretan karakter yang sama secara berurutan lebih dari tiga karakter pada contoh di atas adalah 100001 jika ada, maka kita dapat melakukan pemampatan/kompresi, kemudian setelah itu beri *bit* penanda pada *File* tersebut, *bit* penanda dapat dipilih secara acak dengan asumsi digunakan secara konsisten dan berupa 8 (delapan) deretan *bit*. *Bit* penanda ini fungsinya adalah untuk menandai, bahwa karakter selanjutnya adalah karakter pemampatan/kompresi sehingga tidak mengalami kesulitan ketika akan mengembalikan *File* yang telah dimampatkan/kompresi ke *File* aslinya, *bit* penanda diatas 111110. Selanjutnya tambahkan deretan *bit* untuk menyatakan jumlah karakter yang berurutan sama seperti pada contah di atas adalah 1000 dan diberi deretan *bit* yang menyatakan karakter yang berulang, yaitu 00010. Dengan demikian dari contoh diatas dapat menghemat *Bandwich* sebanyak 5 *bit*.

2. Algoritma Dekompresi (Decompression) dengan Metode Run-Length

Apabila kita memerlukan data yang telah dimampatkan/kompresi, data tersebut dapat dikembalikan pada bentuk aslinya tanpa merubah isi yang ada didalamnya, dengan melakukan proses dekompresi terhadap data tersebut. Seperti yang sudah dijelaskan pada pemampatan dengan metode *Run-Lenght*, bahwa pada proses ini pula harus memberikan *bit* penanda pada setiap *File* yang akan dimampatkan sehingga kita dapat mengembalikan bentuk *File* yang telah dimampatkan/kompresi ke bentuk sebelumnya. Adapun langkah-langkah yang harus kita lakukan dalam proses Dekompresi adalah sebagai berikut:

- Melihat karakter pada hasil pemampatan secara berurutan mulai dari awal sampai akhir, jika

ditemukan *bit* penanda, maka proses Dekompresi dapat dilakukan.

- Melihat karakter setelah *bit* penanda, kemudian konversikan ke bilangan desimal untuk menentukan jumlah karakter yang berurutan.
- Melihat karakter berikutnya lalu lakukan penulisan karakter tersebut sebanyak bilangan yang telah diperoleh pada karakter sebelumnya.

Oleh karena itu pada saat akan melakukan proses pemampatan/kompresi dengan menggunakanAlgoritma *Run-Length* harus memperhatikan dalam pemilihan *bit* penanda. *Bit* penanda sebaiknya dipilih pada karakter yang paling sedikit jumlahnya yang terdapat pada *File* yang akan dimampatkan/kompres, sebab jika pada *File* sebalumnya ditemukan karakter yang sama dengan *bit* penanda kemungkinan besar harus menulis karakter tersebut sebanyak dua kali pada *File* pamampatan/kompresi, hal ini bertujuan untuk menghindari kesalahan dalam mengenali apakah *bit* penanda pada *File* pemampatan benar-benar *bit* penanda atau memang karakter dari *File* sebelumnya. Dengan demikian diharapkan dari penjelasan cara kerja dari proses kompresi dengan menggunakan algoritma *Run-Length* pembaca dapat memahami secara garis besar proses tersebut.

3. Algoritma Kompresi (Compressi) dengan Metode Two-Level-Hoshing

Kompresi atau Dekompresi data dapat pula dilakukan dengan menggunakan Algoritma *Two-Level-Hoshing* yang merupakan satu metode dalam proses sistem Kompresi dan Dekompresi data sehingga diharapkan pada sistem Kompresi dan Dekompresi ini dapat memberikan satu solusi bagi *User*(pengguna) ketika menggunakan metode ini, dan dapat pula membandingkan sistem Kompresi dan Dekompresi mana yang lebih efisien dan lebih menghemat media penyimpanan ketika data tersebut telah dikompres tanpa menghilangkan file yang ada seperti telah dijelaskan pada bab sebelumnya, bahwa Algoritma *Two-Level-Hoshing* ini menggunakan metode dengan cara mencari 4 (empat) bit sebelah kiri yang sama.

Pada saat karakter yang 4 (empat) bit pertamanya sama diterima secara berderetan 7 (tujuh) kali atau lebih, maka Algoritma *Two-Level-Hoshing* akan memampatkan data tersebut dengan bit penanda kemudian karakter pertama dari deretan 4 (empat) *bit* terakhir deretan berikutnya dan ditutup dengan *bit* penutup, seperti pada contoh kata " CCCCCCC " berikut :

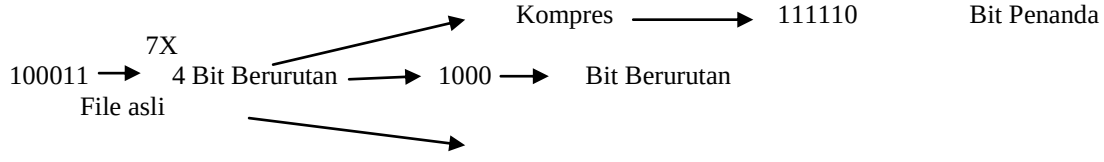
Dik :
Karakter C = 67 (ASCII)
Dalam ASCII C = 67 = 100011
Bit Penanda : 100011

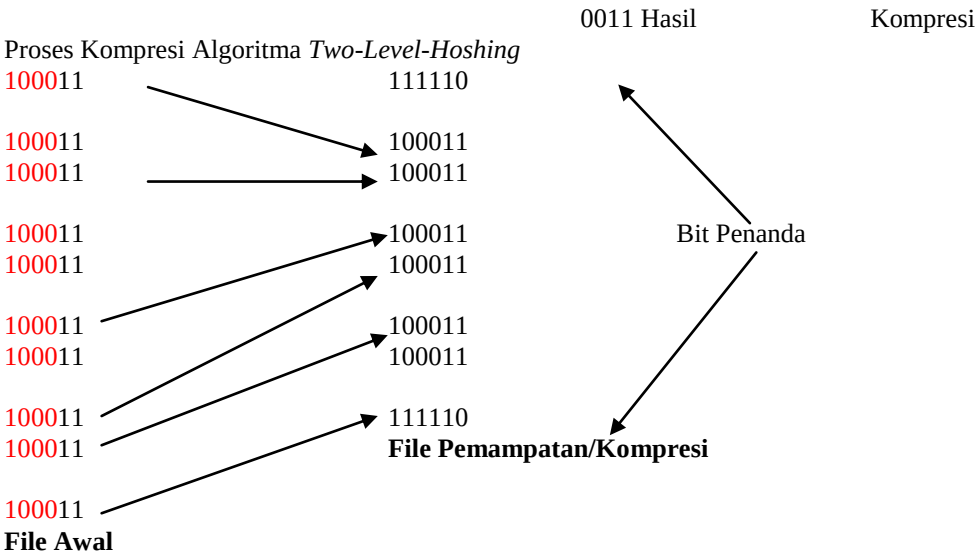
Dit : *kompresi Two-Level-Hoshing*...?

Jwb : cari deretan 4 (empat) bit yang sama dan diikuti 4 (empat) bit Berikutnya yaitu 1000

Karakter = biner.

" CCCCCCC " =1000111000111000111000111000111000111000111100011





Gambar 2. Proses Algoritma Two-Level-Hoshing

Sama seperti pada Algoritma *Run-Lenght* terlebih dahulu yaitu harus melihat apakah terdapat deretan karakter yang 4 *bit* pertamanya sama secara berurutan tujuh karakter atau lebih, jika ada dan memenuhi syarat, maka kita dapat melakukan pemampatan/kompresi terhadap data tersebut. Seperti pada contoh diatas secara berurutan terdapat 10 (sepuluh) karakter yang sama, sehingga pemampatan dapat dilakukan, Berikan bit penanda pada *File* pemampatan, *bit* penanda ini berupa 6 deretan bit yang dapat dipilih secara acak namun digunakan secara konsisten pada seluruh bit penanda pemampatan/kompresi dan pada contoh diatas bit penandanya dipilih 111110, lalu masukan karakter pertama 4 *bit* kiri berurutan dari *File* awal 100011 gabungkan 4 *bit* kanan karakter kedua dan ketiga kemudian tambahkan ke *File* pemampatan, lakukan hal ini terus sampai akhir deretan karakter dengan 4 bit pertama yang sama dan selanjutnya akhiri dengan *bit* penanda pada *File* pemampatan/kompresi.

4. Algoritma Dekompresi dengan Metode Two-Level-Hoshing

Pemampatan dengan menggunakan Algoritma *Two-Level-Hoshing* juga dapat dilakukan pengembalian ke bentuk *File* sebelumnya *(Decompresi)*. Sama seperti pada Algoritma *Run-Lenght* pada Algoritma ini juga harus melakukan pemilihan *bit* penanda agar tidak terjadi kesalahan ketika melakukan Dekompresi. Adapun langkah-langkah yang harus dilakukan adalah:

- Melihat karakter pada hasil pemampatan satu-persatu dari awal sampai akhir, dan jika ditemukan *bit* penanda maka proses Dekompresi dapat dilakukan.
- Melihat karakter setelah penanda, kemudian tambahkan karakter tersebut pada *File* Dekompresi.
- Melihat karakter berikutnya, jika bukan penanda masukan 4 *bit* kanan karakter dibawahnya. Hasil dari gabungan tersebut ditambahkan pada *File* Dekompresi. Lakukan terus sampai ditemukan *bit* penanda.

5. Konteks Diagram

Diagram konteks adalah diagram arus data yang menampilkan batasan dari sistem informasi. *Context diagram* adalah gambaran dari sistem informasi *Top Level Manajemen*. Cara menggambar *context diagram* adalah dengan menempatkan satu simbol proses yang memperlihatkan keseluruhan sistem informasi di tengah halaman dan menggunakan arus data untuk menghubungkan kesatuan luar yang ada ke proses utama

Pada sistem kompresi *File* dengan menggunakan Algoritma *Run-Length* dan *Two-Level-Hoshing* proses sistem pemampatan ini memiliki satu entitas, yaitu *User* (pengguna). *User* atau pengguna adalah pemakai perangkat lunak tersebut yang akan melakukan atau ingin memampatkan/mengkompres *File* yang dimilikinya.

Perangkat lunak tersebut dapat digambarkan dalam sebuah proses

dimana proses tersebut merupakan suatu aktifitas yang besar. Dan untuk menggambarkan sistem kompresi tersebut secara menyeluruh dapat dilihat sebagai berikut:

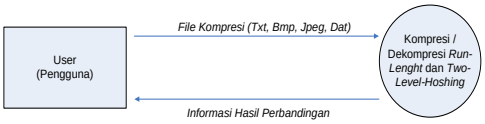


Diagram Konteks
Sistem Kompresi file Run-Length Dengan Two Level-Hoshing

5.1. Diagram Alir Data Level 1

Pada Aliran Data level 1 dapat di bagi kedalam 2 proses, yaitu:

1. Proses *Load*, yaitu *User* mengambil atau membuka *File* yang akan dikompresi atau dekompresi.
2. Proses Kompresi atau Dekompresi, yaitu pemilihan metode yang akan digunakan dimana *user*, atau pemakai akan memilih metode *Run-Length* atau *Two-Level-Hoshing.*

Untuk menggambarkan proses sistem tersebut dapat dilihat pada gambar berikut:

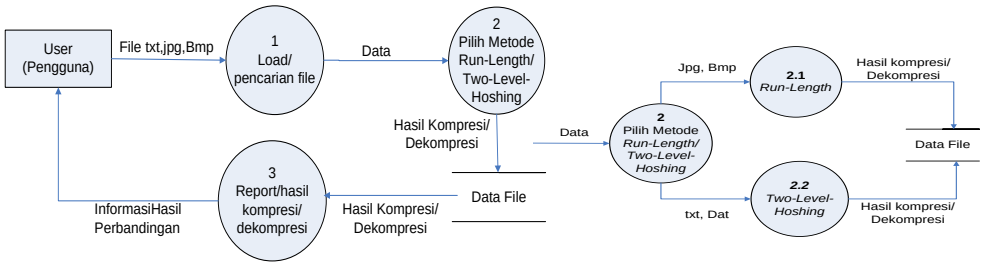


Diagram Alir Data Level 1 Proses Pencarian File

Diagram Alir Data Level 2 Proses Pemilihan Metode Perbandingan

5.2. Diagram Aliran Data Level 2

Proses sistem perangkat lunak kompresi *file* ini dapat dibagi menjadi 2 (dua) buah proses, yaitu:

a. Proses kompresi *file* dengan Algoritma *Run-Length,* dimana pada proses ini format pengenal *file* tersebut ditulis pada *byte* pertama, kedua dan ketiga dengan karakter yang ditentukan, misalnya A, B dan C begitu juga dengan jumlahnya. Karakter berikutnya empat berisi karakter *bit* penanda yang telah ditentukan dengan mencari karakter dengan frekuensi kemunculan terkecil. Karakter kelima dan seterusnya berisi hasil pemampatan dengan Algoritma *Run-Length*. Apabila hasil pemamptan *file* ukurannya masih besar, maka dapat dibandingkan dengan algoritma berikutnya, misalnya *Two-Level-Hoshing.*

b. Proses kompresi *file* dengan Algoritma *Two Level-Hoshing*
Pada kompresi *file* dengan Algoritma *Two-Level-Hoshing* prosesnya sama dengan Algoritma *Run-Length*, yaitu format pengenal *file* ditulis pada *byte* pertama, kedua dan ketiga dengan karakter yang telah ditentukan.jika pada Algoritma *Two-Level_Hoshing file* yang akan dikompres masih berukuran besar, maka dapat dibandingkan dengan Algoritma *Run-Length.*
Kedua proses tersebut dapat digambarkan pada Diagram Alir Data sebagai berikut:

6. Evaluasi Hasil Analisis

Berdasarkan hasil analisis sistem di atas, bahwa ada beberapa kebutuhan sistem yang diperlukan untuk menjadi acuan dalam merancang dan membangun proram ini, adapun kebutuhan-kebutuhan itu adalah, :

1. Kebutuhan *User* (pengguna) akan ruang penyimpanan data yang

besar,sehingga diperlukan suatu proses Kompresi/Dekompresi.

2. Kapasitas *File* yang rata-rata menuntut media penyimpanan yang besar, sehingga *file* tersebut perlu dimampatkan/dikompres.
3. Dibutuhkan suatu metode pemampatan/kompresi yang dapat mengatasi masalah di atas, sehinngga diharapkan dapat memberikan satu solusi yang lebih efektif dan efesien dalam hal penyediaan kapasitas media penyimpanan data, maka dari hasil analisis di atas direkomendasikan adanya pengembangan dari sistem metode pemampatan/kompresi data yang baru seperti berikut :
   a. Metode pemampatan/kompresi yang baru harus dapat mengatasi kebutuhan *User* (pengguna) dalam hal kebutuhan ruang penyimpanan data yang besar.
   b. Metode pemampatan/kompresi yang baru harus dapat memampatkan/mengkompres file yang besar, sehingga dapat menghemat ruang media penyimpanan data.
   c. Dengan menggunakan perangkat lunak Kompresi/Dekompresi metode *Run-Length dan Two-Level-Hoshing* diharapkan dapat mengatasi masalah diatas.

Untuk mencapai tujuan tersebut yang akan menjadi acuan dari perancangan program Kompresi dan Dekompresi baik itu perancangan *Input* atau *Output* , maka rancangan program yang akan dibangun harus mencapai sasaran dan tujuannya. Adapun tujuan dan sasaran dari rancangan program itu adalah sebagai berikut:

1. Rancangan sistem yang akan dibuat dapat membantu *User* (pengguna) dalam proses pemampatan/kompresi file dan juga dalam proses pengembalian bentuk file/dekompresi.
2. Rancangan sistem yang dibangun harus disesuaikan dengan kebutuhan akan media penyimpanan yang ada dengan melalui beberapa proses yang diperlukan seperti Proses Kompresi dan Proses Dekompresi.
3. Rancangan sistem yang dibuat harus lebih mempercepat suatu pekerjaan, sehingga dapat tercapai efisiensi dan efektifitas dalam hal Kompresi dan Dekompresi file

Untuk dapat melakukan Proses Kompresi dengan menggunakan Algoritma *Run-Length* dan *Two-Level-Hoshing*, tentunya kita harus mengetahui bagaimana proses dari kedua Algoritma itu sendiri, sehingga dalam inplementasi kedalam bentuk program nantinya kita tidak mengalami kesulitan, Adapun Algoritma dari kedua metode tersebut adalah sebagai berikut

7. Perancangan Sistem

1. Algoritma Kompresi dengan Metode Run-Length

```
Var
     i,ii : LongInt;
     BitPenanda,inbuf ,buflong,same,ulang4,uu :integer;
     SigmaMin:LongInt;
     Save_ Cursor:TCursor ;
     Sigma Ascii:Array[0..255] of LongInt;
Begin
    For uu:=0 to 255 do             {ulangi selama uu > kecil atau sama dengan
dari  255}
    Begin
       SigmaAscii[uu]:=0             {isi sigma Ascii[uu] dengan 0}
     end ;
     SigmaMin:=1000000;              {isi sigma dengan 1000000}
     For i=0 to Fsize-1 do;   {ulangi selama i lebih kecil atau sma dengan dari
Fsize-1}
     begin;
       Inbuf:=integer(Buf[1+i]);     {isi Inbuf denganinteger (Buf[1+i])}
```

```
       inc(SigmaAscii[Inbuf]);       {isi SigmaAscii[Inbuf] ditambah 1}
       end;
       for uu:=0 to 255 do           {ulangi selama uu lebih kecil atau sama
dengan dari 255}
        begin
       Inbuf:=interger(Buf[1+i]);    {isi Inbuf dengan 10000000}
       Inc(SigmaAscii[Inbuf]);       {isi Sigma Ascii[Inbuf] ditambah 1}
       End;
       For uu:=0 to 255 do         {ulangi selama uu lebih kecil atau sama dengan
dari frize-1}
        Begin ;
       If SigmaAscii[uu]<SigmaMin then      {jika SigmaAscii[uu] lebih kecil dari
SigmaMin}
       Begin
           SigmaMin:= SigmaAscii[uu];       {isi  SigmaMin dengan SigmaAscii[uu]}
           BitPenanda:=uu;                  { isi BitPenanda dengan uu}
       End;
        End;
     BufC[4]:=(BitPenanda);                 { isi BufC[4] dengan chr(BitPenanda)}
     ii:=4;                                 { inisialisasi ii dengan 4}
     i:=1;                                  { inisialisasi ii dengan 1}
     ulang4:=1;                             { inisialisasi ulang4 dengan 1}
     SameBit:=Integer(Buf[i]);              { isi SmaeBit dengan Integer (Buf[i]}

     While i<Fsize do                       {ulangi selama i lebih kecil dari
Fsize}
      Begin
       While iSameBit=BitPenanda do  {ulangi selamaSmaeBit sama
                                     denganBitPenanda}
       Begin
           ii:=ii+1;                        {Tambahkan ii dengan1}
          Bufc[i]:=chr(BitPenanda);         {isi BufC[ii] dengan chr(BitPenanda)}
          ii:=+1;                           {tambahkan ii dengan1}
       Bufc[ii]:=chr(BitPenanda);           {isi BufC[ii] dengan chr(BitPenanda)}
       i:=1+i;                              {tambahakan i dengan1}
       SameBit:=Integer (Buf[i])            {isi dengan Integer(Buf[i]}
                End;
           i:=1+i;                          { tambahkan ii dengan 1}
       BufLon:=Integer(Buf[i});             {isi buflong dengan integer (Buf[i])}
       If  (buflong=SameBit) and (buflong<>BitPenanda) then   {jika Byte
selanjutnya  sama}
          Begin
          If ulang4<255 then ulang4:= 4+1   {jika ulang4 lebih kecil dari 255
maka Ulang 4
                                     ditambah 1}
       else
         begin
             BufC[ii+1]:=chr(BitPenanda);   {isi BufC[ii+1] dengan
chr(BitPenanda)}
              BufC[ii+2]:=chr(ulang4);      {isi Bufc[ii+2] dengan chr(ulang4)}
              BufC[ii+3]:=chr(SameBit);     {isi Bufc[ii+3] dengan chr(SameBit)}
              ii:=ii+3;                     {tambahkan ii dengan 3 )}
               ulang4:=1;                   {isi ulang4 dengan1}
       end;
        end
     else                                   {Byte selanjutnya tidak sama}
        Begin
       If ulang4>3 then                     {jika ulang4 lebih besar dari 3}
          Begin
             If BitPenanda<>ulang4 then     {jika BitPenanda tidak sama
denganulang4}
                 Begin
              BufC[ii+1]:=chr(BitPenanda); {isi BufC[ii+1]dengan chr(BitPenanda)}
              BufC[ii+2]:=chr(ulang4);       {isi Bufc[ii+2]dengan chr (ulang)}
              BufC[ii+3]:=chr(SameBit);      {isi Bufc[ii+3] dengan
chr(SameBit)}
              ii:=ii+3;
           end
           else                            {jika ulang4 lebih kecil atau sama
dengan dari 3}
           begin
                BufC[ii+1]:=chr(BitPenanda);  {isi Bufc[ii+1]dengan
chr(BitPenanda)}
```

```
                  BufC[ii+2]:=chr(ulang4-1);    {isi Bufc[ii+2] dengan chr(ulang4-
1)}
                  BufC[ii+3]:=chr(SameBit);     {isi Bufc[ii+3] denganchr(SameBit)}
                 BufC[ii+4]:=chr(SameBit);      {isi Bufc[ii+4] dengan
chr(SameBit)}
                ii:=+4;
              end;
        end
       else                                     {jika ulang4 lebih kecil atau sama
                dengan dari 3}
        begin
            for uu:=1 to ulang4 do
               begin
                ii:=ii+1;                       {isi ii ditambah 1}
                BufC[ii]:=chr(SameBit);         {isi Buflong sama dengan BitPenanda}
                      ii::=ii+1;                {isi ii ditambah 1}
                BufC[ii]:=chr(BitPenanda)       {isi Bufc[ii] denganchr(BitPenanda)}
                ii:=ii+1;                       {isi ii ditambah 1}
                BufC[ii]:=chr(BitPeananda);     {isi Bufc[ii] dengan
chr(BitPenanda)}
                If i<Fsize then                 {jika i lebih kecil dari Fsize}
                Begin
                    i:=1+i;                    {isi ii ditambah 1}
                    buflong:=Integer(Buf[i];   {isi buflong dengan Integer(Buf[i]}
                end;
                 end;
                 if ulang4>3 then              {jika ulang4 lebih kecil atau sama
dengan dari 3}
                  Else
                 Begin
                If sameBit<>BitPenanda then    {jika sameBit tidak sama dengan
BitPenanda}
                Begin
                    ii:=ii+1;                  {isi ii ditambah 1 }
                    Bufc[ii]:=chrSameBit);     {isi Bufc[ii] dengan chr(SameBit)}
        End;
              End;
        End;
         End;
```

2. Algoritma Kompresi dengan Metode Two-Level-Hoshing

```
Var
      SameTLH,i,ii,uu: LongInt;
    Inbuf, Genap, BitPenanda:Integer;
    BitKiriS, BitKiriT: String;
    BitKanaS, BitKananT: Stirng;
    SigmaMin: LongInt;
    jadiPenanda:Boolean;
    SigmaAscii:Array[0..255] of LongInt;
Begin
    for uu:= 0 to 255 do              {ulangi selama uu lebih kecil atau sama
dengan dari 255}
    begin
        SigmaAscii[uu]:=0;            {isi SigmaAscii dengan 0 }
    End;
    SigmaMin:=10000000;               {isi SigmaMin dengan 10000000 }
    For i:=0 to Fsize-1 do            {ulangi selama i lebih kecil atau sama dengan
dari Fsize-1}
    Begin
        Inbuf:=integer(Buf[1+i);      {isi Inbuf dengan integer(Buf[1+i]}
        Inc(SigmaAscii[Inbuf]; {tambahkan SigmaAscii[Inbuf] dengan 1 }
    End;

    For uu:=0 to 255 do               {ulangi selama uu lebih kecil atau sama
dengandari 255 }
      Begin
        If SigmaAscii[uu]<SigmaMin then {jika SigmaAscii[uu] lebih kecil dari
SigmaMin]}
        Begin
        SigmaMin:=SigmaAscii[uu];     {isi SigmaMin dengan SigmaAscii[uu]}
        BitPenanda:=uu;               {isi BitPenanda dengan uu }
         End;
```

```
          End;
        BufC[4]:=chr(BitPenanda);     {isi BufC[4] dengan chr(BitPenanda)}
        ii:=4;                        {isi ii dengan 4 }
        BitKiriS:=(inttohex(Integer(Buf[Fsize]),2)); {isi BitKiriS dengan
hexadecimal
                                                     dari
integer(Buf[Fsize]),2)}
        Delete(BitKirirS,2,1);        {hapus BitKiriS}
        If BitKiriS='4' then Buf[Fsize+1]:='P' else Buf[Fsize+1]:='O';
                                      {jika BitKiriS sama dengan  '4' maka isi
                                      Buf[Fsize+1] dengan'P', jika tidak isi
Buf[Fsize+1]dengan'O'}
        i:=1;                         {isi i dengan 1 }
        SameTLH:=1;                   {isi SameTLH dengan 1 }
        JadiPenanda:=false;           {isi JadiPenanda dengan false}
        BitKiriS:=(inttohex(integer(Buf[i]),2));     { isi BitKiriS dengan
hexadecimal  Dari
                                                     integer(Buf[i])}
        Delete(BitKiriS,2,1);         {hapus BitKiriS}
        While i<=Fsize do             {ulangi selama i lebih kecil atau sama dengan
dari Fsize}
        Begin
            If (SamaTLH=1) and (Buf[i]=chr(BitPenanda)) then JadiPenanda:=True;
                                      {jika SameTLH=1 dan Buf[i]=chr(BitPenanda)
                               maka
                                        isi JadaPenanda dengan true}
          BitKanaS:=(inttohex(integer(Buf[i],2));    {isi KananS dengan hexadecimal
                                                     dari integer (Buf[i]))}
          Delete(BitKananS,1,1);      {hapus BitKanaS}
          I:=i+1;                     {tambahkan i dengan1 }
          BitKiriT:=(inttohex(Integer(Buf[i]),2));   {isi BitKiriS dengan
hexadecimal
                                                     dari Integer(Buf[i])}
           Delete(BitKiriT,2,1);      {Hapus BitKiriS}
        If (BitKiriT=BitKiriS) and (JadiPenanda=false) then {jika BitKiriT sama
                                                             dengan BitKiriS dan
                                                             JadPenanda sama
                                                             dengan false}
           begin
                SameTLH:=SameTLH+1;   {tambahkan SameTLH dengan BitKiri dan
                                                JadiPenanda sama dengan false}
                If (SameTLH mod 2)=1 then     {jika Hasil bagi SameTLH dengan 2
sama
                                                dengan 1 maka...}
                begin
                    Bit KananT:=(inttohex(Integer(Buf[i],2)) {isi BitKananT dengan
                                                             hexadecimal dari
                                                             Integer(Buf[i])}
                  Delete(BitKananT,1,1);      {Hapus BitKananT}
                  If StrtoIntDef('S'+BitKananT,255)   {isi BitKananT dengan
                                                       hexadecimal dari
Integer(Buf[i]}
          JadiPenanda:=True;          {jika
StrtoIntDef('S'+BitKananT,255)=BitPenanda
                                        maka jadiPenanda sama dengan true}
                  End;
                        End;
                        Else                        {jika tidak sama}
                  Begin
                  i:=i-SameTLH;{kurangi i dengan SameTLH}
                  if (SameTLH<7) or (jadiPenanda=true) then {jika SameTLH lebih
kecil dari 7 atau
                                                             JadiPenanda sama
                dengan true}
                  begin
                     for uu:=i to i+SameTLH-1 do      {ulangi selama uu lebih kecil
atau
                                                       sama dengan dari i+SameTLH-1
}
                  begin
                     ii:=ii+1;        {tambahkan ii dengan 1 }
                  BufC[ii]:=Buf[uu];  {isi Buf[ii] dengan Buf[uu]}
                  If Buf[uu]=chr(BitPenanda) then     {jika BufC[ii] sama dengan
                  chr(BitPenanda)}
```

```
                begin
                   ii:=ii=+1;        {tambahkan ii dengan 1 }
                  BufC[ii]:=Buf[uu]; {isi BufC[ii] dengan Buf[uu}}
                       End;
                    End;
               End
               Else                          {jika SameTLH>=7 }
          Begin
                    ii:=ii+1;                {tambahkan ii dengan 1 }
             BufC[ii]:=chr(BitPenanda);      {isi BufC[ii] dengan chr(BitPenanda)}
             ii:=ii+1;                {tambahkan ii dengan 1 }
             BufC[ii]:=Buf[i];        {iai BufC[ii] dengan Buf[i]}
             Genap:=1;        {iai genap dengan 1 }
             For uu:=i+1 to i+SameTLH-1 do {ulangi selama uu=i+1 lebih kecil atau
                                            sama dengan i+SameTLH-1 }
             Begin
                  Case genap of
                  1:                 {jika genap = 1 }
             Begin
             Bit KiriS:=(inttohex(Integer(Buf[uu],2));      {isi BitKiriS dengan
                                                            hexadecimal dari
                                                            Integer(Buf[uu])}
                 Delete(BitKiriS,1,1);        {hapus BitKiriS}
                 Genap:=0;                    {isi genap dengan 0 }
             End;
             0:                               {jika genap 0 }
             Begin
                 Bit KiriT:=(inttohex(Integer(Buf[uu]),2)); {isi BitKiriT dengan
                                                             hexadecimal dari
                                                             Integer(Buf[uu])}
                 Delete(BitKiriT,1,1);        {hapus BitKiriT}
                 ii:=ii+1;                    {tambahkan ii dengan 1 }
                 BufC[ii]:=chr(StrtoIntDef('S'+BitKiriS+BitKiriT,255));
                 {isi Bufc[ii]
                 Denganchr(StrtoIntDef('S'+BitKiriS+BitKiriT,255))}
                 Genap:=1;                    {isi genap dengan 1 }
                        End;
                    End;
                End;
           ii:=ii+1;                          {tambahkan ii dengan 1}
           BufC[ii]:=chr(BitPenanda);{isi Bufc[ii] (dengan chr(BitPenanda)}
           If genap=0 then                    {jika genap sama dengan 0 }
           Begin
               ii:=ii+1;                      {tambahkan ii dengan 1 }
               BufC[ii]:=chr(Integer(Buf[i+SameTLH-1]));    {isi Bufc[ii] dengan
                                                  chr(Integer(Buf[i+SameTLH-1}
           end;
               end;
         i:=i+SameTLH;               {tambahkan i dengan SameTLH}
         SameTLH:=1;                 {isi SameTLH dengan 1 }
         JadiPenanda:=false;                {isi jadiPenanda dengan false}
         BitKiriS:=(inttohex(Integer(Buf[i]),2));   {isi BitKiriS dengan
hexadecimal
                                                    dari
                                                   Integer(Buf[i]}
         Delete(BitKiriS,2,1);               {hapus BitKiriS}
            End;
          End;
      End;
```

3. Algoritma Dekompresi dengan Metode Run-Length

```
Var
      i,ii:LongInt;
    BitMin,uu, KarakterLDO, BitPenanda:Integer;
    JumSama:Integer;
    BitKiri, BitKanan, KodeTem, KodeBim:String;
    DapatKode:Boolean;
    KarakterHuf:Array[0..255] of Integer;
    JumlahKarakterHuf: Array[0..255] of Integer;
Begin
    I:=0;                            {inisialisasi i dengan 0}
```

```
    BitPenanda:=Integer(Buf[4];       {inisialisasi BitPenanda dengan
Integer(Buf[4])}
    ii:=4;                                  {inisialisasi ii dengan 4 }

    Repeat                            {ulangi sampai ii>=Ukuran file (FileSize)}
        inc(ii);                            {tambahkan ii dengan 1 }
      if Integer(Buf[ii]=BitPenanda then    {isi Bufc[i] dengan Ascii dari
BitPenanda}
      begin
      if Integer(Buf[ii+1])=BitPenanda then  {jika Buf[ii]=BitPenanda}
      begin
      inc(i);                         {tambahkan i dengan 1}
      BufC[i]:=chr(BitPenanda);       {isi BufC[i] dengan Ascii dari BitPenanda}
          inc(i);                     {tambahkan ii dengan 1 }
      end
     else
      begin
          JumSama:=Integer(Buf[ii+1]); {isi JumSama dengan Integer(Buf[ii+1])}
          For uu:=1 to Jumsama do     {ulangi selama uu=1 lebih kecil atau sama
                                      dengan dari JumSama}

          Begin
             inc(i);                  {tambahkan i dengan 1 }
            Buf[i]:=Buf[ii+2];        {isi BufC[i] dengan Buf[ii+2]}
          End;
             Ii:=2;                   {tambahkan i dengan 1 }
         End;
    End
    Else                              {jika Buf[ii]<> BitPenanda bararti Karakter
                                      Asli}
  Begin
      inc(i);                         {tambahkan i dengan 1 }
      BufC[i]:=Buf[ii];               {isi BufC[i] dengan Buf[ii]}
     End;                             {akhir dari pengulangan}
Until ii>=Fsize;
End;
```

4. Algoritma Dekompresi dengan Metode Two-Level-Hoshing

```
Var
  i,ii:LongInt;
  BitMin, uu, KarakterLDO, BitPenanda: Integer;
  JumSama:Integer;
  BitKiri, BitKanan, KodeTem, KodeBin: String
  DapatKode: Boolean;
  KarakterHuf: Array[0..255] of Integer;
  JumlaKarakterHuf:Array[0..255] of integer;
Begin
      {inisialisasi}
      i=0;                            {inisialisasi i dengan 0}
      BitPenanda:=Integer(Buf[4];     {inisialisasi BitPenanda dengan
Integer(Buf[4])}
      ii:=4;                          {inisialisasi ii dengan 4 }
Repeat                                {ulangi sampai ii lebih besar atau sama
dengan ukuran
                                        file (Fsize)}
inc(ii);                              {tambahkan ii dengan 1 }
if Integer(Buf[ii])=BitPenanda then   {jika Buf[ii+1] sama dengan BitPenanda}
begin
  if Integer(Buf[ii+1])=BitPenanda then      {jika Buf[ii+1] sama dengan
BitPenanda}
begin
  inc(i);                             {tambahkan i dengan 1 }
  BufC[i]:=chr(BitPenanda);           {isi Bufc[i] dengan Ascii dari BitPenanda}
  ind(ii);                                {tambahkan ii dengan 1 }
end
else                                  {jika Buf[ii+1] tidak sama dengan BitPenanda}
begin
  inc(i);                             {tambahkan i dengan 1 }
  BufC[i]:=Buf[ii+1];                 {isi BufC[i] dengan Buf[ii+2]}
  BitKiri:=(inttohex(integer(Buf[ii+1),2));  {isi BitKiri dengan hexadecimal dari
Buf[ii+1]
                                          dengan dua digit}
  Delete(BitKiri,2,1);                {hapus BitKiri}
```

```
  ii:=ii+2;                          {tambahkan ii dengan 2 }
  while (integer(Buf[ii])<>BitPenanda) and(ii<Fsize) do     {ulangi sampai
                                       Buf[ii]<>BitPenanda dan (ii<Fsize)}

Begin
  BitKanan:=(inttohex(integer(Buf[ii]),2));  {iai BitKanan dengan hexadecimal dari
Buf[ii]
                                       dengan dua digit}
  Delete(BitKana,2,1);        {hapus BitKanan}
  inc(i);                            {tambahkan i dengan 1 }
  BufC[i]:=chr(StrtoIntDef('$'+BitKiri+BitKanan,255));      {isi BufC[i] dengan
Ascii
                                                    Dari
                                                ('$'+BitKiri+BitKanan,255)}
  BitKanan:=(inttohex(Integer(Buf[ii],2));   {isi BitKanan dengan dengan
hexadecimal
                                         dari Buf[ii] dengan dua digit}
  Delete(BitKanan,1,1);              {hapus (bitKanan)}
  inc(i);                            {tambahkan i dengan 1 }
  BufC[i]:=chr(StrtoIntdef('$'+BitKiri+BitKanan,255));      {isi BufC[i] dengan
                              Dari tdef('$'+BitKiri+BitKanan,255))}
          inc(ii);                                  {tambahkan ii dengan 1 }
         end;
      end;
end
else                                 {jika karakter asli}
begin
    inc(i);                                {tambahkan i dengan 1 }
    BufC[i]:=Buf[ii];                {isi Bufc[i] dengan Buf[ii]}
    End;
Until ii>=Fsize                      {akhir dari pengulangan}

End;
         End;
```

KESIMPULAN

Setelah menyusun, mempelajari dan menganalisa metode algoritma *Run_Length dan Two_Level_Hoshing* untuk proses kompresi dan dekompresi dapat disimpulkan bahwa proses kompresi dan dekompresi file dapat membantu pengguna (*user)* dalam hal proses penyimpanan *file* yang besar ke dalam ukuran yang relatif kecil sehingga tidak membutuhkan media penyimpanan yang besar.

Dari hasil perancangan dan implementasi metode algoritma *Run_Length* dan *Two_Level_Hoshing* untuk sistem kompresi dan dekompresi, maka dapat diambil beberapa kesimpulan, yaitu sebagai berikut :

1. Algoritma *Run_Length* dan *Two_Level_Hoshing* dapat dipergunakan dalam sistem kompresi dan dekompresi.
2. Tidak ada algoritma yang paling efektif untuk setiap file karena hasil kompresi setiap algoritma tergantung dari isi file yang akan dikompres.
3. Dari kedua metode yang digunakan dalam sistem kompresi file yaitu algoritma *Run-Length* lebih efektif mengkopres file yang berekstensi Jpeg dan Bmp, sedangkan algoritma *Two-Level-Hoshing* lebih efektif mengkopres file yang berekstensi Text, Doc dan Dat. Hal ini disebabkan masing-masing algoritma memiliki kelebihan dan kekurangan

DAFTAR PUSTAKA

Adam Osborne, Davis Bunnel dan Ir. Setiyo Utomo, "*Pengantar Komputer Mikro",* Jakarta : Erlangga, 1986

Addison-Wesley Publishing Company, "International Computer Science Series".

"Compression file with Two Level Hoshing Algorithim" (www.science Computing. Com)

Fathansyah, Ir, "Basis Data", Bandung : Informatika, 2002

G.H Gonnet, Handbook of Algorithms and Data Structure, London,

Jogiyanto H,M.,Kadir, Abdul, Ir., M.T, (2001), "Pemograman File Base Menggunakan Delphi 6.0", Jakarta, Penerbit Salemba Infotext.

Rinaldi Munir, Ir dan Leoni Lidya, Ir, "Algoritma dan Pemograman", Bandung: Informatika 1997

Roger S. Pressman, Ph.D, "Rekayasa Perangkat Lunak", Yogyakarta : Andi, 2002

Tachbir Hendro P.S.Si, M.T, "Modul kuliah Algoritma dan Logika" , Bandung, 1999